Un algorithme d'election entierement distribue et decentralise dans un systeme distribue quelconque

C. Lavault

INRIA

CENTRE DE ROCQUENCOURT

Rapports de Recherche

N° 625

UN ALGORITHME D'ÉLECTION ENTIÈREMENT DISTRIBUÉ ET DÉCENTRALISÉ DANS UN SYSTÈME DISTRIBUÉ QUELCONQUE

Christian LAVAULT

Février 1987

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

UN ALGORITHME D'ELECTION ENTIEREMENT DISTRIBUE ET DECENTRALISE DANS UN SYSTEME DISTRIBUE QUELCONQUE

A FULLY DISTRIBUTED DECENTRALIZED ELECTRON ALGORITHM IN AN ARBITRARY ANONYMOUS DISTRIBUTED SYSTEM

Christian LAVAULT
Université Paris 12
58, avenue Didier
94210 LA VARENNE

et

INRIA
Domaine de Voluceau
Rocquencourt - BP. 105
78153 LE CHESNAY

ABSTRACT

The design and analysis of a performing fully distributed decentralized random algorithm for determination of a spanning tree and election in an arbitrary anonymous network topology (symmetry breaking problem) are completed herein. Its advantages consist in a real weak corpus of assumptions, in the techniques involved, and in the respective lower and upper bounds of $0(n\log n)$ and $0(n^2)$ number of mesages required. No particular assumptions is actually formulated upon the topology of the underlying graph, but connectivity. Every site (or process) in the network is assumed to be absolutely anonymous. Moreover, the initial knowledge remains only local. In this very sense, this asynchronous random algorithm may be considered fully distributed and decentralized, and thereby, very different from most of distributed algorithms designed so far.

RESUME.

Ce travail présente l'évolution et l'analyse des performances d'un algorithme d'élection probabiliste entièrement distribuée et décentralisé dans un système distribué anonyme et asynchrone quelconque. Son intérêt réside dans sa grande généralité, dans les techniques qu'il utilise et dans les performances obtenues.
Aucune hypothèse particulière n'est formulée sur la topologie du réseau (sinon sa connexité) ; les sites (ou processus) de celui-ci sont, au départ, absolument anonymes ; de plus, aucun processus n'a besoin de connaître ni la structure globale, ni même la taille du réseau. En ce sens, l'algorithme présenté ici est bien entièrement distribué et décentralisé ; il se distingue des algorithmes habituels qui s'appuient sur une topologie particulière, et parfois une connaissance globale de celle-ci (anneau en maillage complet par exemple).
La méthode d'élection utilisé : la construction d'un arbre couvrant pour le graphe qui modélise le réseau, et sa mise en oeuvre, sont également fructueuses et permettent l'obtention du routage associé très facilement. L'évaluation des performances algorithmiques, enfin, donne de bon résultats ; en particulier, les complexités en moyenne sont calculées pour le nombre de messages émis et le "temps d'exécution de l'algorithme".

SOMMAIRE

6. ANALYSE DE L'ALGORITHME

6.1. Nombre maximal de phases de l'algorithme

6.2. Complexité en messages de l'algorithme

6.2.1. Borne supérieure de la complexité en messages

6.2.2. Borne inférieure de la complexité en messages

6.2.3. Remarques sur la complexité moyenne en messages

6.3. Complexité en temps

6.3.1. Taille moyenne d'un fragment

6.3.2. Complexité en temps attendue dans le pire des cas

BIBLIOGRAPHIE

7. ANNEXES

Remarques sur la complexité en messages de l'algorithme pour des topologies de réseaux particulières

7.1. Anneau uni - et bi-directionnel

7.2. Réseau à maillage complet

7.3. Comparaison avec le présent algorithme

1. INTRODUCTION

1.1. Systèmes distribués et algorithmes distribués

Nous définirons un *système distribué* S comme un réseau modélisé par un graphe $G = (\mathcal{P},L)$ où $|\mathcal{P}| = n$ sites sont reliés par $|L| = e$ lignes de communication directe. Chaque processus sur un site de S possède une mémoire locale non partageable et ne peut communiquer qu'avec ses voisins dans S, et ce, uniquement par l'envoi et la réception de messages.

On peut caractériser le comportement de ces processus sur les sites de S en les assimilant à des automates à états finis au sein d'un système de communication par routage de messages. Autrement dit, chaque processus à un instant donné est dans un état déterminé (parmi un ensemble fini d'états possibles), et, à la réception d'un message, ce processus réalise de façon séquentielle un certain nombre d'opérations primitives dont la nature dépend à la fois de son état courant et du contenu du message reçu. Les opérations primitives possibles sont : des "calculs" locaux (la réalisation d'un programme séquentiel partiel), la transmission de messages, et le changement d'état. On peut ainsi décrire le comportement d'un processus sur un site de S par un schéma de transition du type :

$$< \text{état}_k, \text{ message reçu } > \rightarrow < \text{état}_{k+1}, \text{ message envoyé, "calcul"} >$$

Par conséquent, un *algorithme distribué* sur S n'est autre que la caractérisation des types d'opérations à réaliser séquentiellement par un processus d'un site de S à la réception d'un message, ce processus étant dans un état donné. Afin d'assurer une réalisation d'opérations entièrement distribuée au sein du système S, on supposera que tous les processus sur les sites de S exécutent le même algorithme et possèdent les mêmes définitions de contexte : ils sont "symétriques" (cf. Raynal [7,19] et van Leeuwen et alii [23]).

1.2. Le problème de l'élection

La solution de certains problèmes de contrôle relatifs aux applications et aux systèmes distribués peut passer par la définition d'un rôle particulier que va jouer l'un (et un seul) des sites qui supportent l'application. C'est par exemple le cas dans les bases de données réparties où le problème du contrôle de la concurrence, dû à l'exécution parallèle de transaction accédant à des données distribuées, peut être résolu en associant à chaque ensemble de données (dont chacune est dupliquée sur les divers sites du réseau) un site particulier.

Le rôle de ce site est de contrôler les accès qu'effectuent les transactions de façon à assurer la cohérence mutuelle de ces données (respect des contraintes d'intégrité) et la convergence de leurs copies vers une même valeur [7,19] ; un tel site est alors généralement appelé "site primaire". La désignation d'un processus destiné à coordonner les diverses activités dans un système distribué constitue un autre exemple du même problème [12].

Il est alors essentiel de savoir définir un tel site et de donner à chacun des autres sites les moyens de communiquer avec lui. Ce site privilégié est généralement choisi en fonction d'un critère de poids, et il est d'usage de choisir celui dont le poids est un extremum (en considérant comme "poids d'un site" son identité représentée par un numéro). Lorsque tous les sites ont des identités distinctes, le ***problème de l'élection*** se ramène à donner à chaque site les moyens de reconnaître un site qui est déjà "par essence" particulier de par son identité, et à l'élire sur cette base. Le problème de l'***élection*** n'est donc en fait résolu par un algorithme véritablement décentralisé et distribué que lorsque les sites y sont tous supposés anonymes et sans connaissance globale du réseau.

Dans tous les cas, une fois le choix du site élu effectué, il est nécessaire d'une part de communiquer l'identité de ce site à tous les autres sites et, d'autre part, d'établir des chemins de communication (des structures de contrôles : arbre couvrant, etc) de ces sites vers le site élu.

Une première solution consisterait à effectuer le choix d'un site, à établir les chemins correspondants de manière statique, et à incorporer ces éléments dans les définitions des sites lors de la génération du système ou de l'application. L'absence de souplesse de cette solution est rédhibitoire et a conduit à la recherche de solutions dynamiques.

Ces solutions réalisent ce qu'il est convenu d'appeler un algorithme distribué d'***élection***. Désormais, nous utiliserons indifféremment les termes de "sites", "processus" et "sommets de G" ; en effet, un site se présente comme une activité unique du point de vue de l'élection, et se modélise dans G par un sommet. Tous les sites/processus sont donc initialement candidats à jouer le rôle particulier d'élu, et, au terme de l'algorithme, seul celui dont l'identité est un extremum peut effectivement le jouer. Si les sites sont supposés anonymes, deux identités, maximale et minimale, sont créées de toute façon au cours de l'algorithme non-déterministe lui-même.

Par conséquent, avant le début du processus d'élection, tous les sites du réseau sont dans un même état (dit ***initial***) ; au début du processus, l'état des processus actifs est également commun, mais différent (il est dit ***candidat***) ; à la fin du processus d'élection, tous les sites sont à nouveau dans le même état (dit ***battu***), à l'exception d'un seul et unique qui se trouve être dans un autre état, distinct de l'état battu (et dit ***élu***).

1.3. Travaux connexes

Le ***problème de l'élection*** a été étudié de façon intensive ces dernières années sous l'hypothèse d'identités distinctes des sites du réseau. Dans ce cas, il s'agit alors d'un problème de calcul d'extremum, et diverses bornes (inférieures et supérieures) en ont été établies dans différents contextes et sous des hypothèses multiples : selon la topologie du réseau (anneaux ou maillages complets le plus souvent) ; selon l'information sur la topologie globale du réseau qui est mise à la disposition locale des sites (connaissance de la topologie particulière du réseau, ou seulement de sa taille, ou connaissance seulement de l'orientation dans un réseau complet par exemple, etc) ; selon que le réseau lui-même est synchrone ou asynchrone, etc.

Sous l'hypothèse de réseaux synchrones, Frederickson & Lynch d'un côté et Vitanyi de l'autre ont montré indépendamment que, dans un anneau, O(n) messages suffisent pour élire un site ; cependant, le nombre de tours synchrones nécessaires (i.e. unités de temps) est une fonction exponentielle des identités : $O(n.2^i)$, où i est l'identité du site finalement élu. Cette borne en temps a été ultérieurement réduite par Gafni jusqu'à $O(n.2^n + i^2)$ (cf. [23]). Comme dans tous ces résultats, chaque message contient au moins l'identité d'un site, le nombre total de bits utilisés dans ces algorithmes est un O(nlgi)*). Récemment, Overmars & Santoro ont conçu un algorithme d'élection synchrone qui sur un anneau nécessite seulement O(nlgn) bits et O(n.i) unités de temps. (cf. [23]).

Si l'on ne s'astreint pas à l'hypothèse d'unicité de chaque identité de sites (soit à cause de l'absence d'identité, soit parce que les identités existent, mais ne sont pas toutes distinctes), il est clair que le ***problème de l'élection*** d'un site ne peut être résolu par une méthode de ***calcul d'extremum***. Plus encore, Angluin montre dans [1] que, si les sites sont tous anonymes (ou, de manière équivalente, possèdent tous une identité commune), il n'existe alors aucune solution déterministe au ***problème de l'élection*** distribuée, que le réseau soit synchrone ou asynchrone **). Par conséquent, toute solution au problème ne saurait être obtenu dans ce cas que par un algorithme non-déterministe.

---

*) note (1) Ici et dans toute la suite, $\lg x = \log_2 x$ $(x \in]0, +\infty[)$.

) note (2) De là la "nouvelle" appellation du problème : **"Symmetry Breaking Problem". Celle-ci met l'accent sur le choix - déterministe ou non - d'un site particulier, indépendemment des poids des sites dans le réseau : il s'agit de briser la symétrie c'est à dire la similitude des comportements et des attributions des sites au sein d'un système distribué quelconque (ici un réseau anonyme asynchrone) - (voir partie 2.2).

A l'exception de l'algorithme récent de Raynal et alii [7], où aucune hypothèse particulière n'est formulée sur la topologie du réseau (sinon sa connexité), tous les algorithmes d'***élection*** développés jusqu'à présent s'appuient sur des hypothèses fortes concernant le système distribué. Souvent d'abord, on présuppose que tout site possède une certaine connaissance globale, plus ou moins importante, du réseau ; ce qui élimine le problème de l'établissment des chemins ou de structure de contrôle entre le site élu et les autres. Ensuite, le maillage du réseau est pratiquement toujours restreint aux deux types classiques : l'anneau (uni ou bi-directionnel [2,4,9,17,18,21]) ou le maillage complet [8,13,20] (avec parfois, en sus, la connaissance par les sites de l'orientation du graphe associé au réseau [13,20]).

Les meilleures performances obtenues pour de tels algorithmes sont en O(nlgn) pour la complexité en messages : moins de 3/4 n.Hn en moyenne sur un anneau bidirectionnel en [2], et, de même, des bornes inférieures et supérieures en O(nlgn) dans un réseau complet sans autre hypothèse, en [8]. Par contre, en [13,20], dans un réseau complet avec connaissance de l'orientation, la complexité en messages atteinte est O(n). ces algorithmes ont, dans des réseaux synchrones ou "synchronisables", une complexité en temps de O(lgn) en général (pour la "synchronisation" de tels algorithems à l'aide de "phases", voir la partie 6.). Enfin, le seul algorithme totalement distribué de Raynal et alii [7] a une complexité en messages de $O(n^2)$ dans le pire des cas et de O(n) dans le cas le plus favorable, dans un réseau absolument quelconque.

Le travail qui suit résoud le ***problème de l'élection*** distribuée, ou ***Symmetry breaking problem***, dans un réseau anonyme et asynchrone quelconque grâce à un algorithme probabiliste entièrement distribué et décentralisé. Cet algorithme construit une arborescence couvrante du graphe G associé au réseau ; sa racine est l'un des sites du réseau : le site élu. L'arborescence couvrante constitue alors une véritable structure de contrôle permettant à tous les processus de communiquer, via des chemins uniques, avec le processus associé à la racine. L'algorithme est de conception très proche de celui de l'arbre couvrant minimal de Lavallée & Roucairol proposé en [11].

Le plan est divisé en six grandes parties. Dans la seconde, on y précise les hypothèses en détail et l'intérêt qu'elles présentent ; la troisième partie expose les principes généraux qui sous-tendent l'algorithme ; la quatrième partie présente l'algorithme lui-même, sa description informelle d'abord, puis son écriture en langage "pseudo-C.SP." ; la cinquième partie est consacrée à la preuve de l'algorithme et la sixième à l'analyse de ses performances : la complexité en messages dans le cas le plus

favorable, le plus défavorable, et en moyenne; la complexité en temps (ou nombre de "phases" de l'algorithme considérées comme unité de temps - voir la partie 6.) est calculée également dans les trois cas classiques (meilleur, pire, et en moyenne). La septième partie conclut en synthétisant les résultats obtenus. Après la bibliographie, l'annexe enfin étudie le comportement de l'algorithme dans deux réseaux particuliers typiques : l'anneau (uni - et - directionnel) et le maillage complet.

2. HYPOTHESES

2.1. Hypothèses sur le réseau de communication

Les processus ne peuvent s'échanger de l'information qu'à l'aide de messages véhiculés sur le réseau ; il n'y a donc aucune mémoire partagée entre les processus.

Le réseau de communication est modélisé par un graphe connexe (sinon, il existerait des sous-réseaux ne communicant pas entre eux) absolument quelconque. Les liaisons entre deux processus sont bidirectionnelles ; en d'autres termes, le graphe est simple et symétrique (ou "non-orienté", et sans boucle ni arête multiple). Ces liaisons sont reliées par le protocole de communication suivant (cf. [4,6,10,14]) :

(i) Les messages ne se perdent pas ; tout message envoyé d'un processus P_i à un processus P_j finit par être reçu par P_j au bout d'un temps arbitrairement long, mais fini ;

(ii) Le contenu des messages n'est ni modifié ni altéré par le routage ;

(iii) L'ordre d'envoi des messages est supposé préservé. Les messages envoyés de P_i à P_j (voisins dans le réseau) atteignent le tampon de réception de P_j dans l'ordre de l'envoi ; autrement dit, par hypothèse, il ne peut y avoir de dépassement de messages sur les lignes ;

(iv) Si deux messages atteignent simultanément le tampon de réception d'un processus, ils y sont ordonnés arbitrairement et intégrés dans cet ordre au tampon (ce dernier est supposé de longueur non bornée pour plus de facilité - et sans perte de généralité (cf. [14]).

2.2. Hypothèses sur les processus

Le réseau distribué asynchrone comporte n processus anonymes. Initialement, les processus n'ont aucune identité et, chaque processus ne connaît que ses voisins directs ; au

cours de l'algorithme, il n'apprend d'eux que leur identité, (déterminée par un tirage au sort sur l'intervalle entier [1,*maxint*] ; *maxint* représentant le plus grand entier positif "possible" calculable par la machine.

Au cours du calcul distribué, un processus quelconque ne peut jamais connaître la structure globale du réseau, pas même le nombre total n de sites ; sa connaissance topologique se limitera toujours à ses propres portes, i.e. : l'identité de ses voisins directs. Au sujet de la connaissance du nombre total n de sites du réseau, il convient cependant de noter que, si n reste inconnu pour tout site, tout site en a cependant une connaissance indirecte dans le tirage aléatoire de son identité, numéro entier compris entre 1 et maxint. Comme il est souligné en [23] par Van Leeuwen et alii, le ***symmetry breaking problem*** peut être résolu par un algorithme probabiliste si on fait conjointement l'hypothèse de symétrie des sites et de leur connaissance de n (nombre total de sites).

Dans le présent algorithme *), la seconde hypothèses n'est que "partiellement" vérifiée, ou plutôt indirectement par la connaissance de *maxint* qui est un majorant de n. Le ***symmetry breaking problem*** est ici résolu dans un réseau distribué anonyme quelconque par un algorithme probabiliste sans l'hypothèse de "symétrie des sites" et de la connaissance d'un majorant donné de n : *maxint* (lié et fixé par la machine). Si n > *maxint* par exemple, l'algorithme d'élection est impossible.

2.3. Intérêt de telles hypothèses

Ces hypothèses sont particulièrement fructueuses car elles facilitent énormément la résistence aux pannes du réseau et les problèmes posés par la régénération de systèmes distribués quelconques. En effet, comme la définition d'un site au sein d'un système distribué anonyme n'inclut aucune détermination d'identité (du fait de l'anonymat) ni d'information globale d'aucune sorte, la configuration du système peut être facilement modifiée : aucune reconfiguration n'entraîne de nouvelle définition de sites (anonymat) et le système lui-même n'est tributaire d'aucune topologie particulière. En fait, en cas de nécessité de reconfiguration locale (du fait de panne ou autre), seuls les processus dont le voisinage a été modifié doivent être redéfinis et régérés (tout en conservant leur anonymat initial).

---

*) Note : Tous les processus ont les mêmes définitions de contexte et exécutent les mêmes texte d'algorithme : en ce sens, l'algorithme (et les processus) sont bien symétriques (cf. Raynal [7,19]).

Comme un processus ne connaît son voisinage que par l'identité des liaisons (canaux) qui le connectent à ses voisins, l'identité de ceux-ci peut être connue grâce à un protocole simple [22] : deux messages par canal (un dans chaque sens) permettent l'échange des identités entre tout couple de processus voisins (une fois des identités distinctes acquises).

3. PRINCIPES GENERAUX DE L'ALGORITHME

Dans un contexte séquentiel, une réponse au problème consisterait à rechercher, dans le graphe associé au réseau, le sommet possédant la plus grande identité, puis à construire, avec ce processus pour racine, une arborescence couvrante du graphe qui établirait des chemins entre tout processus et la racine.

Dans un contexte distribué, la solution consistant à faire envoyer par tous les processus leur identité à l'un d'entre eux est impossible car - outre le fait qu'elle suppose des identités distinctes et ne respecte pas les hypothèses de localité de l'information - elle nécessite la définition préalable d'un processus unique qui collecterait toutes les information (absence de symétrie). Ce qui reviendrait à résoudre préalablement un problème ... d'***élection***.

Le présent algorithme est fondé sur quatre principes généraux (ou règles). Si l'on met à part le premier d'entre eux et, bien sûr, ce qui est conséquence de l'anonymat (identités non-distinctes a priori), ces principes sont très semblables à ceux mis en oeuvre dans l'algorithme de [11].

Une des caractéristiques essentielles du calcul distribué réside dans le fait qu'un algorithme qui produit un résultat unique peut être démarré simultanément par un, plusieurs, voire tous les processus. Le résultats quant à lui doit, a priori, rester indépendant du nombre de processus qui ont effectivement activé le début de l'algorithme.

Initialement, tous les processus sont dans un même état, dit état ***initial***. Tout processus se trouvant dans cet état peut spontanément devenir ***actif*** à tout moment ; étant anonyme, sa première tâche avant la réalisation de toute autre opération, est alors - dans l'état ***actif*** - de se donner une identité par génération d'un nombre pseudo-aléatoire qui sera cette identité : un numéro entre 1 et ***maxint***. Les identités des processus sont donc obtenues par tirages aléatoires indépendants d'une distribution uniforme sur l'intervalle

d'entiers [1,*maxint*]. Par conséquent, une fois activés et à l'issu de ce tirage aléatoire de leurs identités, plusieurs processus peuvent avoir des identités égales entre 1 et *maxint* ; ceci est une caractéristique fondamentale de l'algorithme (liée à son caractère non-déterministe) et lui confère sa généralité théorique et pratique.

Cette première étape d'activation-dénomination de processus constitue donc le point de démarrage de l'algorithme ; ces processus passent alors dans l'état *candidat*.

Dès la deuxième étape donc, un ou plusieurs processus peuvent être simultanément *candidats*. La deuxième étape met en oeuvre le principe général fondamental de l'algorithme : la construction d'une arborescence couvrante du graphe G associé au réseau. Cette construction s'effectue par la mise en place de sous-arbres du graphe G, enracinés et orientés (de la racine vers les sommets pendants), appelés fragments de G. Comme l'algorithme peut être démarré par un ou plusieurs processus en même temps, un ou plusieurs fragments peuvent être simultanément construits sur G. Ces fragments se constituent par l'envoi de messages par des processus candidats (éventuellement un seul au départ) vers d'autres processus candidats (éventuellement activé(s) par le premier). Les processus en question entrent alors dans la procédure centrale de "négociation - connexion" (procédure NEGOCIATION) qu'ils mènent à bien (simultanément) jusqu'à l'acceptation où le rejet des propositions de connexion (voir [11]) selon des règles précisées plus loin.

L'originalité principale de ces règles et de la méthode de construction de l'arborescence couvrante de G tient au fait que, à l'inverse des autres algorithmes distribués du même type conçus jusqu'à présent ([5,8,13,20]) où un fragment cherche à ce qu'un autre se connecte à lui, cherche à en absorber un autre, ici, comme en [11], un fragment est candidat à se fusionner à un autre : les demandes de connexion dans l'algorithme se font donc, comme dans [11], dans le sens inverse des demandes de connexion des algorithmes de [5,8] ou [20] *). L'autre caractéristique est aussi que les identités des processus sont, dans le présent algorithme, comparées selon la relation d'ordre total non strict existant entre elles (du fait de l'existence d'identités non-distinctes), au lieu d'une relation d'ordre total strict dans les autres algorithmes ([5,8,11,13,20]) ; les réponses aux demandes de connexion de processus à processus (ou de

*) C'est pourquoi à la différence des algorithmes de [6] et [8], il n'est pas nécessaire ici de mettre à jour toute l'information locale à chaque sommet du nouveau fragment : la mise à jour de la seule identité suffit. De plus, tout fragment peut se fusionner à tout autre sans l'attente d'un "niveau" égal commun, comme en [6].

fragment à fragment) se font donc selon la relation spécifique du présent contexte. Ceci conduit au troisième principe général qui s'applique durant la deuxième étape de l'algorithme : la procédure TEST.

Dans la mesure où il est possible que plusieurs processus (et, éventuellement tous les processus, avec la probabilité infime de $\frac{1}{(maxint)^n}$) puisse tirer aléatoirement le même numéro d'identité entre 1 et maxint, le processus de négociation-connexion peut échouer du fait de conflits entre racines de certains fragments créés, et/ou la terminaison correcte de l'algorithme peut en être affectée. Dans le cas où les racines d'au moins deux fragments ont la même identité (conjointement avec tous les processus appartenant respectivement à chacun d'eux), il s'agit de mettre en oeuvre également la terminaison correcte de l'algorithme distribué. Il s'agit en particulier de détecter les cycles potentiels et de s'affranchir d'éventuels deadlocks. La procédure TEST permet de résoudre ces problèmes par la réalisation d'autant de nouveaux tirages aléatoires de numéros d'identité qu'il est nécessaire (voir la partie 4.1.1.).

La dernière règle concerne la terminaison correcte de l'algorithme. Lorsque tout processus sans exception a pu être correctement connecté à son père, "l'apprentissage de la terminaison" peut avoir lieu, depuis les feuilles de l'arborescence couvrante jusqu'à l'unique sommet - racine. Le processus élu (l'unique racine restante) peut alors envoyer tout le long de l'arborescence un message de terminaison de l'algorithme qui prévient tous les sommets.

4. L'ALGORITHME ET SA SPECIFICATION

4.1. Description informelle de l'algorithme

Durant toute l'exécution de l'algorithme, chaque sommet du réseau est soit la racine d'un fragment quelconque de G (sous-arbre présenté et enraciné de G), soit un simple sommet non-racine au sein d'un fragment qu'il détermine (état "*candidat*") ; au cours de l'exécution de l'algorithme, "l'identité d'un fragment" n'est autre que celle de sa racine.

4.1.1. La procédure NEGOCIATION et la procédure TEST

Dans tout ce qui suit, on se place du point de vue d'un processus quelconque que l'on nommera P_i.

P_i maintient en mémoire les variables locales suivantes :

- dem_i est la variable booléenne qui détermine si P_i a envoyé un message nego ou non ;

- $ident_i$ est la variable à valeurs dans {1,...,maxint} représentant le numéro d'identité, ou identité, de P_i ;

- rac_i est la variable booléenne déterminant si le sommet P_i est ou non une racine (rac_i = 0 ou 1) ;

- $état_i$ prend les valeurs **initial, candidat, battu** ou **élu** et, quand l'algorithme est terminé, la proposition suivante est vraie : $\exists!j\ \forall\ k\neq j$ ($état_j$=***élu*** $\wedge$ $état_k$=***battu***) (jusqu'à la terminaison, la proposition suivante est vraie : $\forall\ P_i$ ($root_i$=1 $\wedge$ $état_i$=***candidat***) $\vee$ ($root_i$=0 $\wedge$ $état_i$=***battu***) ; voir la partie 4.1.3) ;

- $pred_i$ est la variable à valeurs entières qui nomme le sommet prédecesseur de P_i dans le fragment auquel il appartient ;

- $fils_i$ et $fils\_sort_i$ sont, respectivement, l'ensemble des successeurs de P_i dans son fragment (initialisé à $\emptyset$), et l'ensemble (initialisé à $\{P_i\}$) des sommets P_x, du fragment F_i tels que l'arête (P_x,P_y) est "sortante" (i.e. : $P_y \notin F_i$, mais $P_x \in F_i$) ; par définition alors, $P_x \in fils\_sort_i$ et la "porte" de P_x, (P_x,P_y) est une "porte sortante".

- $idnum_x$ est une variable à valeurs entières particulière contenue dans les messages d'acquit testacq (messages de type acq).

$fils\_sort_i$ et $idnum_i$ sont toutes deux utilisées au sein de la procédure TEST (voir la définition de $idnum_x$ dans la procédure TEST). Le calcul d'élection distribué est présumé terminé lorsque $idnum_x$ a été décrémentée jusqu'à zéro, pour tout P_x (sommet sortant du fragment de l'unique racine restante), et que l'ensemble $fils\_sort_i$ est vide.

Tout fragment disponible F_i peut envoyer un message à tout autre fragment F_j par l'intermédiaire de leurs racines respectives P_i et P_j. Ce message est une demande de P_i à P_j afin de savoir si, selon la relation d'ordre existant entre les identités ($ident_i < ident_j$, ou $ident_i > ident_j$, ou $ident_i = ident_j$), P_i peut ou non se connecter à P_j : autrement dit, si F_j peut ou non absorber F_i. La procédure NEGOCIATION peut être exécutée simultanément à un moment donné par plusieurs racines disponibles, et par conséquent déclancher l'absorption de certains des fragments correspondants. De la même

manière, toute racine disponible P_i peut envoyer un message à tout autre sommet disponible non-racine P_j, et appeler ainsi la procédure NEGOCIATION. P_j peut donc finir par devenir la nouvelle racine du fragment F_i : ce dernier prend alors l'identité de sa nouvelle racine et devient un sous-fragment de F_j. Cette procédure peut être aussi appelée simultanément par plusieurs racines disponibles.

L'algorithme n'exécute la procédure TEST que dans le cas très particulier où plusieurs (au moins deux) identités de processus se trouvent être égales (éventuellement toutes, avec une probabilité très faible). Il s'agit alors de briser le cercle vicieux causé par l'existence d'identités non toutes distinctes, de manière à réaliser des négociations quasi-normales et des connexions correctes deux à deux entre fragments "identiques". C'est ainsi que lorsqu'un sommet quelconque P_i (racine ou non) reçoit un message de demande de négociation de la part d'une racine quelconque dans le réseau possédant la même identité, la procédure TEST est appelée. Celle-ci procède alors en deux étapes distinctes.

Tout d'abord, le sommet P_i renvoie un message de refus (***nok***) à la racine qui demande la négociation, puis il diffuse à tous ses fils un message ***test*** demandant des messages réponse ***testacq*** (message d'acquis définis dans la partie 4.1.3) de la part de tous les fils de P_i possédant une porte sortante non encore utilisée. Chacun de ces messages ***testacq*** contient en effet la valeur de la variable *idnum* correspondant à un fils sortant : pour tout fils sortant de P_i, P_x, $idnum_x$ = (nombre total de portes de P_x) - (nombre de portes de P_x d'identité égale à $ident_x$). Si (P_x,P_y) est une porte sortante de P_x (arête sortante d'extrémité P_y), l'identité de la porte sortante en question est l'identité de l'extrémité de l'arête : $ident_y$; si donc P_i est racine du sommet sortant P_x, $ident_x = ident_i$ et toute porte sortante de P_x d'identité égale à $ident_x$ est telle que $ident_y = ident_x = ident_i$.

Ensuite, le processus dépend désormais entièrement des valeurs de $idnum_x$ pour les P_x correspondants. Tant qu'il reste encore un fils sortant P_x de P_i possédant une arête sortante non-utilisée tel que $idnum_x \neq 0$, alors, on abandonne TEST (jusqu'à la terminaison au plus) et P_x envoie un message de demande de négociation (***nego***) à P_y, appelant ainsi la procédure NEGOCIATION. Dans son déroulement, l'algorithme reviendra inévitablement à TEST lors de la terminaison. A contrario, lorsque chaque fils sortant P_x de P_i est tel que $idnum_x = 0$, alors P_i doit changer d'identité et effectue un retirage aléatoire uniforme dans [2,***maxint***]. La nouvelle identité de P_i est en effet calculée par la formule $ident_i := |\ ident_i$ - aléa ([2,***maxint***])| + 1, ce qui est équivalent à effectuer un tirage aléatoire uniforme dans l'ensemble $\{1,...,maxint\} \setminus \{ident_i\}$.

Désormais, $état_i$ = ***candidat***, et aussi longtemps que l'ensemble $fils\_sort_i$ reste non vide, la procédure TEST est réexécutée récursivement. En sorte que P_i change d'identité à chaque passage dans TEST et, ce faisant, devient éventuellement un sommet simple, non-racine, au cours de quelque exécution de TEST où $(\exists\ P_x)\ idnum_x \neq 0$. Ce processus récursif continue de réduire l'ensemble fil_sort_i jusqu'à ce que celui-ci finisse par être vide, avec $(\forall\ P_x)\ idnum_x = 0$. C'est alors que l'unique racine restante est dans l'état ***élu***, et que l'algorithme entame son processus de terminaison effective : cette racine restante unique diffuse un message fin à tous les sommets du réseau (son fils dans son fragment) et l'algorithme s'arrête.

4.1.2. L'état d'un processus racine

Tout sommet peut envoyer ou recevoir des messages, mais seuls les processus racines sont capables de créer des messages : un sommet simple, non -racine, n'émet un message qu'en réponse à un message reçu au préalable. Soit P_i un sommet racine du fragment F_i, l'objectif fondamental de P_i est de tenter de fusionner son fragment F_i à tout autre fragment (éventuellement composé d'un seul sommet) et, ce faisant, de devenir lui-même un sommet simple, non racine.

Ainsi donc, toute racine disponible P_i peut, de façon non-déterministe, soit envoyer ou recevoir un message de demande de négociation (**nego**) en liaison avec tout autre sommet disponible quelconque P_j, soit recevoir un message de réponse (**ok**, **nok**) à un précédent message de négociation envoyé par P_i. Dans le premier cas, à la réception d'un message **nego** de demande de négociation de la part d'un sommet P_j, la racine P_i compare sa propre identité, $ident_i$ à β, le numéro d'identité contenu dans le message. Selon la relation d'ordre entre β et $ident_i$, la racine P_i accepte, refuse, ou appelle la procédure TEST (dans le cas où $ident_i = \beta$) : respectivement, P_i renvoie à P_j un message **ok**, un message **nok**, ou bien entre dans l'exécution de TEST. Dans le second cas, lorsque P_j répond à P_i par un message ok, P_i diffuse à tous ses fils dans le fragment un message **nouv_rac** avec l'instruction de mettre à jour leurs identités, à $ident_j$.

Remarque : Lorsqu'une racine P_i reçoit un message réponse **nok** de la part d'un sommet quelconque P_j, P_i peut démarrer une nouvelle procédure de négociation. Il faut noter qu'entre temps, P_j peut éventuellement avoir envoyé un message de demande de négociation à quelque autre sommet. A la réception d'un message **nouv_rac**, tout racine P_i met à jour son propre fragment selon l'identité de la nouvelle racine.

4.1.3. L'état d'un processus non-racine et les messages

Aucun sommet simple, non-racine, ne peut démarrer seul une procédure de négociation ; il ne peut que répondre à la mise en oeuvre d'une telle procédure. Soit P_i un sommet non-racine dans un fragment F_i, une fois la proposition ($rac_i = 1 \wedge état_i =$ ***candidat***) non satisfaite, elle ne le sera plus jamais : on aura toujours ($rac_i=0 \wedge état_i=$***battu***).

Lorsque le message reçu par P_i est :

- **nego** (demande de négociation-connexion) : deux cas sont possibles. Si P_i n'est pas le sommet destinaire du message, P_i se contente d'assurer sa transmission à ses fils dans F_i. Si P_i est le sommet destinataire, il se comporte normalement comme une racine à la réception d'un tel message (cf. partie précédente).

- **ok** (acceptation de connexion) : P_i transmet le message à son prédécesseur dans F_i, lequel fait de même, ... et ainsi de suite jusqu'à ce que le message atteigne la racine de F_i. Après quoi, P_i attend un message **nouv_rac**.

- **nok** (rejet de connexion) : le message est simplement transmis de P_i à son prédécesseur dans F_i.

- **nouv_rac** (réalisation de connexion) : P_i met à jour son identité selon l'identité de la nouvelle racine de son fragment.

- **test** (utilisé dans la procédure TEST) : le message est diffusé par une racine quelconque à tous ses fils, de manière que chaque fils sortant dont une porte reste à essayer puisse renvoyer à sa racine son propre message **testacq** (voir parties 4.1.1 et 4.3).

- **testacq** (message d'acquit dans TEST) : il s'agit du mesage de réponse au message **test** de la racine renvoyé par chaque fils sortant P_x dont une porte est restée inutilisée ; **testacq** contient la valeur de la variable $idnum_x$ correspondante (cf. partie 4.1.1).

- **fin** (terminaison de l'algorithme) : au sein de la procédure TEST, l'unique racine restante finit par diffuser un message **fin** à tous les somemts du réseau et la terminaison de l'algorithme s'effectue.

Remarque : La plupart des messages dont il est ici question, ainsi que les messages réponse induits, restent valables pour les sommets racines. Les messages test et testacq sont reçus et envoyés uniquement dans la procédure TEST.

4.2. Primitives de communication et structure des messages

L'écriture proprement dits de l'algorithme utilise une syntaxe proche du langage C.S.P. de la manière suivante. On convient que lorsqu'un processus quelconque P_i envoie à un autre processus quelconque P_j le message $Port_j$!! <a,b,c> et que, de son côté, le processus P_j reçoit de P_i le message $Port_i$?? <d,e,f>, la signification en est que, dans le processus P_j, d prend la valeur qu'avait a au sein du processus P_i, e celle qu'avait b, etc. Notons que le passage des paramètres se fait comme en FORTRAN, seule compte la place occupée par un paramètre dans la liste constituant le message, et que $Port_j$!! est une primitive non bloquante, à la différence de C.S.P.

(a) Le message structurel, ou plus simplement message, possède quatre composantes ou paramètres, par exemple $Port_i$?? $<\alpha,\beta,\gamma>$ où i est l'identité du processus transmetteur immédiat, α est le contenu propre du message (ok,nok,etc.), β est l'identité du processus transmetteur initial, et γ est l'identité du processus auquel est transmis le message.

(b) Le message d'acquit, ou acquit, est de la forme <acq, μ>, où μ est un entier positif et "acq" a pour contenu *testacq*, message d'acquit au message de demande diffusant test à l'intérieur de la procédure TEST. Les "acqs" permettent de mettre à jour les variables locales *idnum* et sont renvoyés à la racine dès réception du message de demande correspondant, si nécessaire (cf. 4.1.1., 4.1.3., 4.3 pour une définition exhaustive). L'utilisation de ces différents messages est définie dans la partie précédente 4.1.3.

4.3. L'algorithme (en pseudo-CSP)

P_i:: *[$état_i$ = *initial* → [Port??<α,β,γ> → [$état_i$ = *actif* ∧ INITIALISATION]]]
INITIALISATION
$ident_i$:= aléa ([1...maxint])
$état_i$:= ***candidat*** ; dem_i := 0
rac_i := 1
$pred_i$:= nil
$fils_i$:= ϕ
$fils\_sort_i$:= {i}

```
   * [rac_i = 1 ∧ état_i = candidat ∧ dem_i = 0 → [Port_x!!<nego,ident_i,->]]
□
procedure NEGOCIATION

*[∀j Port_j??<α,β,γ> → [root_i=| ∧ state_i=candidat
                  → [α=nego → [β<ident_i → Port_j!!<ok,ident_i,.>; fils_i := fils_i ∪ {j}
                                □
                              β=ident_i → procédure TEST
                                □
                              β>ident_i → Port_j!!<nok,.>; Port_j!!<nego,ident_i,i>
                             ]
                     □
                    α=ok → rac_i := o ∧ état_i := battu ; ident_i := β; pred_i := j; fils_i := fils_i ∪ {j};
                                                  (∀x ∈ fils_i) Port_x!!<nouv_rac,β,.>
                     □
                    α=nok → procédure NEGOCIATION
                    ]
                        □
                     rac_i=o ∧ état_i=battu → [α=nego → [β<ident_i → Port_j!!<ok,ident_i,.>;
                                                                  fils_i := fils_i ∪ {j}
                                                        □
                                                      β=ident_i → Port_pred_i!!<α,β,γ>
                                                        □
                                                      β>ident_i → Port_j!!<nok,.>;
                                                                  Port_j!!<nego,ident_i,.>
                                                      ]
                                                 □
                                                α=ok → Port_pred_i!!<ok,β,.>; {pred_i} := j ;fils_i := fils_i
                                                                  ∪ {j} ; ident_i := β
                                                 □
                                                α=nok → Port_pred_i!!<nok,β,.>
                                                 □
                                                α=nouv_rac → ident_i := β; fils_i := fils_i ∪ {j}
                                                 ]
                    ]
          ]
procédure TEST

     [rac_i=1 ∧ état_i=candidat → (∀x∈fils_i) Port_x!!<test,->];fils_sort_i := fils_i avec arête sortante]
    *[Port_j??<α,β,ident_i> → [α=testacq → mise à jour de fils_sort_i ; Port_j!!<nok,->
                                  [idnum_x=o → [ident_i := |ident_i-aléa([2..maxint])|+1;
                                      (∀x∈fils_i) Port_x!!<nouv_rac,ident_i,.>;état_i := candidat];
                                      [fils_sort_i=φ → état_i := élu ; (∀x∈fils_i)Port_x!!<fin,->); STOP
```

```
                                    □
                                fils_sort_i≠φ → procédure TEST
                                ]
                                □
                            idnum_x≠o → Port_x!!<nego,->
                            ]
                            □
                        α=nego → skip
                            □
                        α=ok → skip
                            □
                        α=nok → skip
                        ]
        □
rac_i=o ∧ état_i=battu → Port_j!!<nok,->;
                        [α=test → (∀x∈fils_i) Port_x!!<test,->];
                        [(∀x∈fils_sort_i) idnum_x := |{Port_x}| -| {Port_j}|; Port_pred_x !!<testacq,idnum_x,i
                          ]
                          □
                        α=testacq → Port_pred_i!!<testacq,idnum_x,->
                          □
                        α=nouv_rac → ident_i := β; (∀x∈fils_i) Port_x!!<nouv_rac,β,->
                          □
                        α=nego → skip
                          □
                        α=fin → STOP
                          □
                        α=ok → skip
                          □
                        α=nok → skip
]
                        ]
]
```

5. PREUVE DE L'ALGORITHME

5.1. Correction partielle

Le Théorème 1. qui suit énonce la propriété de correction partielle de l'algorithme : si cette propriété est vérifiée, alors l'algorithme de construction d'une arborescence couvrante du graphe G présentée dans la partie 4.3. résoud correctement le

problème de l'élection sur un réseau anonyme quelconque - à condition toutefois que cet algorithme distribué vérifie les propriétés de terminaison distribuée adéquates (voir partie 5.2).

Théorème 1. Pour toute exécution de l'algorithme, il existe une racine unique et une seule qui finit par rester : c'est le sommet élu.

Démonstration : Nous allons raisonner par l'absurde. Supposons que le théorème soit faux, comme il est alors impossible d'élire un sommet, c'est à dire d'obtenir une racine unique et une seule qui reste, pour une exécution de l'algorithme, l'hypothèse (H) suivante doit être vérifiée :

(H) - Il existe une exécution de l'algorithme telle que $p > 1$ sommets restent jusqu'au bout des sommets racines ("éternellement" dans l'état **candidat**). Notons alors ces p racines rac_1, rac_2, ..., rac_p et supposons que l'on ait : $(\forall\ i < j)\ ident_i < ident_j$.

La démonstration par l'absurde nécessite cinq lemmes ; le cas où tous les sommets du réseau ont la même identité se ramène en fait au cas général d'identités toutes distinctes par le biais des deux premiers lemmes. Les trois derniers lemmes permettent, dans le cas général où les identités des sommets sont toutes distinctes, de dériver une contradiction avec (H).

Notons que dans cette partie et dans les suivantes, la notion synchrone de "phase" de l'algorithme est introduite. Le concept de phase de cet algorithme (en "unité de temps" dans l'exécution du présent algorithme) permet de synchroniser ce dernier : l'introduction de la notion de "temps" est en effet nécessaire, non seulement pour l'analyse des complexités en temps de l'algorithme, mais à la fois dans la partie faisant la preuve de l'algorithme et dans l'analyse des complexités en messages (par l'intermédiaire de la notion de nombre maximum de phase de l'algorithme). On trouvera la définition rigoureuse d'une phase du présent algorithme dans le début de la partie 6.

Supposons donc que, dans le cas où les sommets du réseau ont toutes la même identité, le Théorème 1 soit faux. Comme à nouveau, il est alors impossible d'élire un sommet, c'est à dire d'obtenir une racine unique et une seule qui reste, pour une exécution de l'algorithme, on doit avoir l'hypothèse (H') suivante :

(H') - Il existe une exécution de l'algorithme telle que p > 1 sommets restent jusqu'au bout des sommets racines. Notons alors ces p racines rac_1, rac_2, ... rac_p et supposons que l'on ait ($\forall$ i,j) $ident_i = ident_j$ avant le démarrage de toute négociation. On peut alors montrer le

Lemme 1. Sous l'hypothèse (H'), il existe une exécution de l'algorithme qui crée au moins p > 1 identités de racines distinctes.

Démonstration : Considérons rac_i ($1 \leq i \leq p$) dont l'$état_i$ est **candidat** et supposons que rac_i reçoive son premier message **nego**, dans la toute première phase de l'algorithme, venant d'une autre racine quelconque de même identité. L'appel puis l'exécution de la procédure TEST va contraindre rac_i à effectuer un retirage aléatoire uniforme dans {1,...,**maxint**} - {$ident_i$} de son identité ; ainsi, la nouvelle identité de rac_i est différente de la précédente. Si rac_i est "vainqueur" dans toutes les négociations qui peuvent avoir lieu au cours de l'algorithme, il est clair que le Théorème 1 est vrai, puisqu'alors rac_i est la seule et unique racine restante (on fait l'unique racine ayant existé durant toute l'exécution de l'algorithme), et donc le sommet **élu** (ce cas est bien spécifique, il ne concerne qu'une exécution particulière de l'algorithme). Sinon, il existe une exécution de l'algorithme dans laquelle la procédure récursive TEST finit par créer au moins p > 1 identités distinctes de p racines dans le réseau, de retirage en retirage. □

Lemme 2. Lorsque tous les sommets ont une même identité, l'hypothèse (H') est en fait équivalente à l'hypothèse (H).

Démonstration : Dans ce cas, sous l'hypohtèse (H'), le Lemme 2 est une conséquence immédiate du Lemme 1. Il existe une exécution de l'algorithme telle que, dans une phase quelconque, il reste jusqu'au bout p > 1 racines, par exemple : rac_1, rac_2, ..., rac_p. Sans perte de généralité, on peut supposer de plus que pour tout i < j, $ident_i < ident_j$ (d'après la démonstration du Lemme 1). L'hypothèse ci-dessus n'est autre que (H). □

Corollaire : Si on suppose que le Théorème 1 est faux, alors, dans tous les cas, l'hypothèse (H) doit être vérifiée.

Démonstration : immédiate. □

On peut désormais démontrer le Théorème 1 grâce à trois lemmes.

Lemmes 3. Sous l'hypothèse (H), tout sommet du réseau doit prendre, dans une phase quelconque, l'identité $ident_i$ (pour un certain i, $1 \leq i \leq p$).

Démonstration : Raisonnons par l'absurde. Si le lemme est faux, il existe au moins un sommet P_j tel que $ident_j \neq ident_i$ durant tout l'algorithme, i.e. : jusqu'au bout. $ident_j$ est forcément une identité que le sommet P_j a reçu d'une racine P_q quelconque qui est maintenant un sommet simple, non-racine (P_q peut être égal à P_j), car sinon $ident_j = ident_i$ pour un certain $1 \leq i \leq p$. Or l'identité de P_q a précisément changé lorsque P_q est devenu un sommet non-racine ; à ce moment, P_q a reçu un message ok et a diffusé un message **nouv_rac** dans son fragment. Ce message contenait l'identité de la nouvelle racine et a forcément atteint P_j. Comme il est clair que P_q n'est jamais redevenu racine, ceci contredit l'hypothèse que $ident_j = ident_q$, pour un certain $q \neq i$, tout au long de l'algorithme. □

Lemme 4. Sous l'hypothèse (H), il existe au moins une racine rac_i (pour un certain i, $1 \leq i \leq p$) qui finit par envoyer un message **nego** vers un sommet extérieur à son propre fragment.

Démonstration : Supposons que rac_i envoie son premier mesage **nego**, dans une certaine phase de l'algorithme, à un sommet P_j. Si $ident_j = ident_i$, le sommet P_j n'étant pas alors dans le fragment de rac_i, le Lemme 4 est vérifié. Sinon, $ident_j = ident_j$; d'après le Lemme 3, le sommet P_j sait que son identité et celle de rac_i sont égales, et la procédure TEST de l'algorithme est appelée. Dans la mesure où $idnum_j = 0$, le sommet P_j change d'identité (par tirage aléatoire dans {1,...,**maxint**} - {$ident_j$}) et rac_i peut alors envoyer un message **nego** vers un sommet sortant quelconque P_x dont au moins une porte est restée inutilisée ($fils\_sort_i \neq 0$). Ce processus d'envoi vers un sommet sortant de porte non utilisée qui est inhérent à la procédure TEST va continuer avec les appels-récursifs à TEST : il existe donc au moins une phase de l'algorithme dans laquelle un message **nego** finit bien par être envoyé en dehors du fragment de rac_i. □

Lemme 5. Supposons que rac_j (pour un certain j, $1 \leq j \leq p$) envoie un message **nego** à un sommet P_α dans le fragment de rac_p ($ident_j < ident_p$). Alors, rac_j deviendra inévitablement un sommet non racine dans le fragment de rac_p, dans une phase quelconque.

Démonstration : D'après le Lemme 4, rac_j ($1 \leq j \leq p$) finit par envoyer un message nego vers l'extérieur de son fragment. Supposons que rac_j envoie ce message nego à un sommet

P_α dans le fragment de rac_p. Si P_α=rac_p, rac_j devient alors un sommet simple, non-racine, du fragment de rac_p, immédiatement après réception du message **ok** de rac_p et les instructions de mise à jour ad hoc. Sinon, le message **nego** en question est réceptionné par P_α qui renvoie un message **ok** à rac_j. Dans chaque cas, on est certain que tout message **ok** envoyé en réponse à un message **nego** provenant d'un sommet d'identité inférieure atteindra rac_j. Alors, rac_j deviendra forcément un sommet simple, non-racine, du fragment de rac_p. □

Il est clair que le Lemme 5 est une contradiction avec l'hypothèse (H) que $p > 1$. Ceci achève la démonstration du Théorème 1. □

5.2. Apprentissage de la terminaison

Théorème 2. L'unique racine restante, dans une certaine phase de l'algorithme, finit par annoncer son **élection**, et l'algorithme s'arrête.

Démonstration : Il s'agit là d'un simple corollaire du Théorème 1 ; en ce sens que, par une démonstration semblable à celle du Lemme 4, on montre aisément que la racine en question finira toujours par épuiser toutes les possibilités de recherche des portes appartenant à ses sommets sortants (*fils_sort*). Ceci se produit par l'envoi de messages **nego** et soit réceptions de messages **ok** (lorsque son identité est supérieure à celle du sommet qui demande à négocier), soit exécution récursive de la procédure TEST tant que les identités correspondantes restent égales ($idnum_x = 0$) et qu'il reste une porte inutilisée dans un sommet sortant ($fils\_sort = \phi$). Lorsqu'il n'y a plus aucune porte inutilisée dans l'ensemble des fils sortants de la racine (i.e. : $fils\_sort = \phi$), celle-ci envoie un message **fin** à chaque sommet du réseau, et chacun des sommets, à la réception du message fin, s'arrête et arrête l'algorithme. □

Ce processus constitue ce qu'on pourrait appeler l'apprentissage de la terminaison de l'algorithme par le réseau.

Noter que, si toutes les identités de sommets sont égales, lorsqu'une racine quelconque "gagne" toutes les négociations où elle est engagée, et ce tout au long de l'algorithme, cette racine reste bien entendu unique et est élue de façon évidente. Cependant, ce processus ne constitue qu'une exécution particulière de l'algorithme et par suite, ne démontre pas le Théorème 1.

6. ANALYSE DE L'ALGORITHME

Cette dernière partie est consacrée à l'étude de la complexité de l'algorithme. La notion de "temps" étant introduite dans les calculs qui suivent sous la forme de phase de l'algorithme - considérée comme unité de temps liée à cet algorithme -, il faut donc commencer par donner une définition rigoureuse de ce concept synchrone. En effet, l'introduction de la notion de temps dans les algorithmes distribués asynchrones pose un problème délicat ; l'utilisation d'intervalles de temps unitaires Θ dans [10] permettait de résoudre théoriquement ce problème de façon tout à fait générale *). Ici, nous allons supposer l'algorithme distribué synchronisé, ce qui revient à se placer dans le cas le plus défavorable (que les identités soient toutes distinctes ou non) :

Définition : La phase ℓ du présent algorithme est caractérisée de la façon suivante : chacun des fragments issus de la phase ℓ-1 émet un message **nego** et un seul. Suivant la relation d'ordre, l'arc correspondant est créé ou non ; la création d'un arc entraînant la concaténation des deux fragments qu'il relie.

La phase ℓ est considérée comme terminée lorsque toutes les concaténations consécutives aux envois de messages **nego** ont été effectuées ; les processus appartenant aux fragments issus de la phase ℓ-1 ont alors changé d'état, conformément à la définition d'un intervalle Θ de [10].

6.1. Nombre maximum de phases de l'algorithme

Lemme 6. Si le réseau distribué possède n sommets avec n identités distinctes, alors, à l'arrêt de l'algorithme, le nombre maximum de phases réalisées est au plus $\lceil \lg n \rceil$.

Démonstration : Au début de la toute première phase de l'algorithme, il existe n sommets anonymes ; à l'issue du processus "d'activation-dénomination", les n sommets ont des identités toutes distinctes. A la fin de la première phase, il reste $\lceil n/2 \rceil$ racines au plus.

---

*) note. Plus précisément en [10] un "intervalle de temps unitaire (ou observable)" était défini comme étant "un intervalle de temps pendant leqsuel un processus quelconque peut changer d'état". La définition donnée ici d'une phase de l'algorithme recoupe parfaitement la notion de temps unitaire Θ conçue à partir d'un changement d'état d'un processus au cours de l'algorithme.

Pour simplifier les notations, nous appellerons **identité** (ou numéro) d'un fragment le numéro (ou identité) de sa racine, et s'il y a q fragments nous supposerons que ceux-ci sont numérotés de 1 à q. De plus, un arc (F_i,F_j) sera créé entre un fragment F_i et un fragment F_j, si et seulement si

- F_i a envoyé un message **nego** à F_j

et

- $F_i > F_j$ (i.e. : $ident_i > ident_j$)

Supposons donc qu'au début de la phase ℓ il y ait q fragments $F_1, F_2, ..., F_q$. La probabilité pour qu'un fragment F_i envoie un message **nego** (variable aléatoire entière M_j) à un fragment F_j donné à l'avance est $\frac{1}{q-1}$ (F_i ne peut s'envoyer de message à lui-même).

Par conséquent, la probabilité pour que F_i n'envoie pas son message à F_j est $1 - \frac{1}{q-1}$.

Par conséquent, pour j donné, la loi de distribution probabiliste des messages **nego** (loi de la v.a. entière M_j) reçus par F_j venant de fragment F_i tels que $F_i < F_j$ ($ident_i < ident_j$) est une loi binomiale :

$$B(k;q-j,\frac{1}{q-1}) = \binom{q-j}{k} (\frac{1}{q-1})^k.(1 - \frac{1}{q-1})^{q-j-k} ,$$

(probabilité pour F_j de recevoir <u>exactement</u> k messages **nego** M_j venant de fragments F_i vérifiant la condition voulue).

Le nombre moyen de tels messages reçus par F_j est par conséquent $E[M_j] = \frac{q-j}{q-1}$. Ce qui représente donc également le nombre moyen de messages **ok** envoyés par F_j, c'est à dire le nombre d'arcs créés durant la phase ℓ dont l'extrémité est dans F_j.

Le nombre <u>total</u> d'arcs créés dans la phase ℓ est donc $\sum_{j=1}^{q} \frac{q-j}{q-1} = \frac{q}{2}$.

Chaque arc, créé durant la phase ℓ provoquant la concaténation de deux fragments, s'il y a q frgaments au début de la phase ℓ, il y en aura au plus $\lceil \frac{q}{2} \rceil$ au début de la phase $\ell+1$.

Par conséquent, si au départ le graphe considéré contient n sommets (i.e. est d'ordre n), le nombre maximum de phases de l'algorithme est majoré par $\lceil \log n \rceil$. □

Remarque : Pour arriver à un résultat analogue, dans [6] on suppose une croissance équilibrée des fragments. Ici, $E[M_j]$ décroit avec j et nous montre que la croissance des fragments n'est pas équilibrée, ce qui n'empêche pas que le nombre maximum de phases soit $\leq \lceil \log n \rceil$.

Lemme 7. Si le réseau distribué possède n sommets ayant tous la même identité, alors, à l'arrêt de l'algorithme, le nombre maximum de phases réalisées est au plus n.

Démonstration : Au début de la toute première phase de l'algorithme, il existe donc n sommets anonymes. Durant cette première phase, les sommets **activés** (au maximum n) tirent au hasard un numéro identité. Dans le cas le plus défavorable où l'algorithme aura le nombre maximum de phases, toutes les identités des processus activés sont égales et n processus sont actifs : les n sommets ont donc tous la même identité. Toujours durant cette première phase, il existe alors n sommets-fragments qui envoient chacun un message **nego** et, à la fin de la première phase, il reste n racines, car aucune connexion n'a pû se réaliser immédiatement (égalité de toutes les identités). Durant la deuxième phase, toutes les racines ayant reçu un message **nego** exécutent un retirage aléatoire pour obtenir une nouvelle identité et il existe au plus n-1 racines de ce type. Chacune d'entre elles envoie alors un seul et unique message **nego** et, à la fin de la deuxième phase, il reste au plus n-1 racines. Par conséquent, il reste au plus n-2 racines en fin de phase 3, ..., $n-\ell+1$ racines en fin de phase ℓ, pour tout $1 \leq \ell \leq n$. □

6.2. Complexité en messages de l'algorithme

Dans les deux calculs de complexité qui suivent on se place du point de vue du "meilleur cas attendu", puis du "pire des cas attendu" au sens anglo-saxon du terme "attendu". Ceci nous donne respectivement une borne inférieure et une borne supérieure attendues de la complexité en message de l'algorithme.

6.2.1. Borne inférieure de la complexité en messages

Théorème 3. Le nombre de messages (attendu) nécessaire à l'algorithme est minoré par $\frac{11}{2} n + O(1)$.

Démonstration : Le meilleur cas attendu se produit lorsque tous les sommets du réseau sont d'identités distincts à l'issue du processus d'"activation-dénomination". Le Lemme 6 est alors vérifié, et le nombre maximum de phases de l'algorithme est d'au plus $\lceil \lg n \rceil$.

- Durant la phase 1, on doit attendre au moins : n messages **nego**, $\lceil n/2 \rceil$ messages **ok**, $\lfloor n/2 \rfloor$ message **nok**, et $\lfloor n/2 \rfloor$ messages **nouv_rac**.

- Durant la phase ℓ, on doit attendre au moins : $\lfloor n/2^{\ell} \rfloor$ messages **nego**, $\lceil n/2^{\ell+1} \rceil$ messages **ok**, $\lfloor n/2^{\ell+1} \rfloor$ messages **nok**, et $\lfloor n/2^{\ell+1} \rfloor$ messages **nouv_rac**, pour tout ℓ ($1 \leq \ell \leq \lceil \lg n \rceil$).

- Durant la dernière phase, n-1 messages de plus : **test**, **testacq**, et **fin** sont diffusés. En sorte que le nombre total de message attendu est d'au plus $\frac{5}{2} n \sum_{\ell=1}^{\lg n} (1/2^{\ell}) + 3(n-1)$, et donc minoré par $\frac{11}{2} n + O(1)$. □

6.2.2. Borne supérieure de la complexité en message

Théorème 4. Le nombre de messages (attendu) nécessaire à l'algorithme est majoré par $n^2 + O(n)$.

Démonstration : Le pire des cas attendu se produit lorsque tous les sommets du réseau ont la même identité à l'issue de la 1ère partie de la première phase ("activation-dénomination"). Le Lemme 7 est alors vérifié, et le nombre maximum de phases de l'algorithme est d'au plus n.

- Durant la phase 1, on doit attendre au plus : n messages **nego** et **nok**.

- Durant la phase 2, on doit attendre au plus : n-1 messages **nego**, $\lceil \frac{n-1}{2} \rceil$ messages **ok**, $\lceil n/2 \rceil$ messages **nouv_rac**, et $\lfloor \frac{n-1}{2} \rfloor$ messages **nok**, **test** et **testacq** .

- Durant la phase ℓ, on doit attendre au plus : $n-\ell+1$ messages **nego**, $\lceil \frac{2-\ell+1}{2^{\ell-1}} \rceil$ messages **ok**, $\lfloor \frac{n-\ell+1}{2^{\ell-1}} \rfloor$ messages **nok**, **test** et **testacq**, et $\lceil n/2 \rceil$ messages **nouv_rac**, pour tout ℓ ($1 \leq \ell \leq n$).

- Durant la dernière phase, n-1 messages de plus : **test**, **testacq** et **fin** sont diffusés.

En sorte que le nombre total de messages attendu est d'au moins $\frac{1}{2} n(n-1) + \sum_{\ell=1}^{n} (n-\ell+1) + 2 \sum_{\ell=1}^{n} \frac{n-\ell+1}{2^{\ell-1}} + O(n)$, et donc majoré par $n^2 + O(n)$. □

Remarque : Le nombre total de bits par message est au moins O(lgi), où i est la plus petite identité créée au cours de l'exécution de l'algorithme ; par conséquent, le nombre total de bits transmis durant l'algorithme est au moins O(nlgn lg i).

6.2.3. Taille moyenne d'un fragment

Lorsque les sommets du réseau ont des identités toutes distinctes, nous allons essayer d'estimer la taille moyenne d'un fragment de racine P_i à la phase ℓ. A la fin de la première phase, comme il a été vu dans la partie 6.1., le nombre de messages nego envoyés par le sommet P_i est $\frac{n-i}{n-1}$. C'est donc aussi la taille moyenne du fragment de racine P_i.

Dans ce qui suit, nous confondrons, pour tout sommet P_i, $ident_i$, l'identité de P_i, et i, de manière à simplifier les notations : ainsi, $ident_j > ident_i$ s'écrira, sans perte de généralité ni risque de confusion, j > i.

Notons $t_i^{\ell-1}$ la taille moyenne du fragment de racine P_i en fin de phase ℓ-1. On aura alors la relation :

$$t_i^{\ell} = t_i^{\ell-1} + \left(\frac{1}{n-i} \sum_{j=i+1}^{n} t_j^{\ell-1}\right) \cdot \frac{n-i}{n-1}$$

où $\frac{1}{n-i} \sum_{j=i+1}^{n} t_j^{\ell-1}$ est la moyenne des tailles des fragments de racine P_j tel que j > i ($n \geq j > i$), et $\frac{n-i}{n-1}$ la probabilité, pour P_i d'émettre au moins un message ok. D'où la relation

$$(1) \qquad t_i^{\ell} = t_i^{\ell-1} + \frac{1}{n-1} \sum_{j=i+1}^{n} t_j^{\ell-1}$$

avec $t_j^0 = 0$ et $t_j^1 = \frac{n-i}{n-1}$.

La relation (1) permet d'évaluer t_i^{ℓ} à l'aide de la série génératrice double $T(z,u) = \sum_{i,\ell \geq 0} t_i^{\ell} z^i u^{\ell}$ ($z, u \in \mathbb{C}$) où, de manière équivalente, à l'aide de la f.g.s. $T_{\ell}(z) = \sum_{i \geq 0} t_i^{\ell} z^i$.

On obtient ainsi un nombre moyen de messages nego, ok et nok en O(n), et un nombre moyen de messages nouv_rac en O(n.lgn) : donc une complexité moyenne en messages de O(nlgn).

6.2. Complexité en temps de l'algorithme

Théorème 5. Lorsque les identités des n sites du réseau sont toutes distinctes, la complexité en temps dans le pire des cas est en moyenne $O(\sqrt{\pi n}.\lg n)$.

Démonstration : Pour i fixé, cette fonction est croissante avec ℓ, ce qui était prévisible, et pour ℓ fixé elle est décroissante avec i, ce qui confirme ce que laissait prévoir le résultat du §. 6.1..

Si, à chaque étape on prend pour "temps" maximal écoulé le nombre moyen d'arcs franchi par un message **nouv_rac**, (c'est celui qui parcourt le plus "long" chemin) sur l'arbre de taille maximale (i.e. celui de racine P_1), en moyenne, le nombre d'arêtes franchi est égal à la hauteur de l'arbre, (en moyenne, la hauteur d'un arbre à n sommets est $\sqrt{\pi.n}$, (voir de Bruijn, Knuth & Rice- 1972, et Flajolet & Odlyzko, 1982), c'est à dire ici à l'étape ℓ, pour le sous-arbre de racine P_1 :

$$\sqrt{\Pi.}\left(t_1^{\ell-1} + \frac{1}{n-1}\sum_{j=2}^{j=n} t_j^{\ell-1}\right)$$

ce qui est (largement) majoré par $\sqrt{\Pi.\frac{n}{2}}$ qui est la taille moyenne attendue de l'avant dernier fragment de racine P_1.

Par conséquent, on peut écrire que :

La complexité en temps dans le pire des cas moyens attendus s'exprime en : $O(\lg n.\sqrt{\Pi.n})$. □

De même, lorsque les n sites ont tous la même identité, on a le

Théorème 6. Lorsque les n sites du réseau ont la même identité, la complexité en temps dans le pire des cas est en moyenne de $O(n.\sqrt{\Pi.n})$

Démonstration: Immédiate à partir du Théorème 5., avec un nombre de phase maximal égal à n . □

BIBLIOGRAPHIE

[1] D. ANGLUIN, "Local and global properties in networks of processes", Proc. 12th ACM Symp. on Theory of Computing (April 1980), pp.82-93.

[2] H.L. BODLAENDER, J. VAN LEEUWEN, "New Upperbounds for decentralized Extrema-finding in a ring of processors", STACS 86, Lecture Notes in C.S. (Springer Verlag - Jan. 1986), pp. 119-129.

[3] K.M. CHANDY, J. MISRA, "Distributed Computing on Graphs : Shortest Paths Algorithms", Comm. ACM, Vol. 25,11 (Nov. 1982), pp. 833-837.

[4] E.J. CHANG, R. ROBERTS, "An improved algorithm for decentralized extrema-finding in circular configurations of processors", Comm. ACM, Vol.22,5 (May 1979), pp. 281-283.

[5] E.W. DIJKSTRA, C.S. SHOLTEN, "Termination detection for diffusing Computations", Inf. Proc. Letters, Vol. 11,1 (Aug. 1980), pp. 1-4.

[6] R.G. GALLAGER, P.A. HUMBLET, P.M. SPIRA, "A distributed algorithm for minimum-weight spanning trees", ACM Trans. Prog. Lang. and Syst. 5 (1983), pp. 66-77.

[7] J-M. HELARY, A. MADDI, M. RAYNAL, "Calcul distribue d'un extremum et du routage associe dans un reseau quelconque", IRISA PI Nr. 289, (Mars 1986).

[8] E. KORACH, S. MORAN, S. ZAKS, "Tight lower and upper bounds for some distributed algorithms for a complete network of processors", Proc. 3rd ACM Conf. on principles of Distr. Computing (August 1984), pp.199-207.

[9] E. KORACH, D. ROTEM, N. SANTORO, "A probabilistic algorithm for decentralized extrema-finding in a circular configuration of processors", Res. Rep. CS-81-19, Dept. of Comp. Science, Un. of Waterloo (1981).

[10] I. LAVALLEE, C. LAVAULT, "Algorithmique parallele et distribuee", R.R. INRIA Nr 471 (Dec. 1985); and Proc. 1st IW.DAG (Ottawa, Aug. 1985).

[11] I. LAVALLEE, G. ROUCAIROL, "A fully distributed minimum spanning tree algorithm", Information Processing Letters (to appear 1986); and I. LAVALLEE : These d'Etat, Chap.IV, (1986).

[12] G. LE LANN, "Algorithms for Data sharing Systems which use tickets" Proc. 3rd Berkeley Workshop on Distributed Data Base and Computer Network, (August 1978), pp. 259-272.

[13] M.C. LOUI, T.A MATSUSHITA, D.B. WEST, "Election in a complete network with a sense of direction", Inf. Proc. Letters (1986).

[14] J. MISRA, K.M. CHANDY, "Termination detection of diffusing Computation in CSP", ACM Toplas, Vol. 4,1 (Jan. 1982), pp. 37-43.

[15] J. MISRA, "Detecting Termination of distributed Computation using Markers" Proc. 2nd ACM Symp. on Principles of Distr. Comput., (Aug. 1983), pp. 290-294.

[16] J.A. PACHL, E. KORACH, D. ROTEM, "Lower bounds for Distributed Maximum-Finding Algorithms", J. of the ACM, Vol.31,4 (Oct. 1984), pp.905-918.

[17] M. OVERMARS, N. SANTORO, "An improved election algorithm for synchronous rings", preliminary draft, Carleton Un. (March 1986).

[18] G.L. PETERSON, "An O(nlogn) unidirectional algorithm for the Circular Extrema Problem", ACM TOPLAS, Vol.4,4 (Oct. 1982), pp.758-762.

[19] M. RAYNAL, "Algorithmes distribues et Protocoles", Eyrolles (Ed.), Sept. 1985.

[20] J. SACK, N. SANTORO, J. URRUTIA, "O(n) election algorithms in complete graphs with sense of direction", SCS-TR-49, Carleton Univ. (1984).

[21] N. SANTORO, E. KORACH, D. ROTEM, "Decentralized extrema-finding in circular configurations of processors : an improved algorithm", Congr. Numer. 34, (1982), pp.401-412.

[22] A. SEGALL, "Distributed Network Protocols", IEEE Trans. on Inf. Theory, Vol.IT 29,1 (Jan. 1983), pp.23-35.

[23] J. VAN LEEUWEN, N. SANTORO, J. URRUTIA, S. ZAKS, "Guessing games and distributed computations in anonymous networks", preliminary draft (March 1986).

7. ANNEXES

Remarques sur la complexité en messages de l'algorithme pour des topologies de réseaux particulières

Dans ce qui suit, nous nous plaçons dans le cas où les identités des processus sont toutes distinctes.

7.1. Anneau uni - et bi-directionnel

Dans un anneau uni - ou bi-directionnel, la complexité en moyenne d'un algorithme de cette classe est en $\Theta(n \lg n)$. (cf [4], [9], [16], [17], [21], [23]).

Plus précisément, le nombre moyen de messages dans un anneau uni-directionnel est n.Hn, et le nombre moyen de messages dans un anneau bidirectionnel est $< \frac{3}{4}$ n.Hn (ce qui est meilleur, cf. [2]) .

7.2. Réseau à maillage complet

En moyenne, la complexité en messages des algorithmes de cette classe dans un réseau complet est $O(n \lg n)$. [8].

Le pire des cas est en $O(n^2)$, l'algorithme de Gallager et alii, optimal dans le pire des cas (cf. [6] et [23]), étant en $O(e) + O(n \lg n)$ pour un réseau quelconque.

7.3. Comparaison avec le présent algorithme

Les performances du présent algorithme, $O(n \lg n)$ dans le meilleur des cas et en moyenne, et $O(n^2)$ dans le pire des cas pour un réseau quelconque sont donc tout à fait acceptables lorsqu'on l'applique à des topologies de réseaux particulières. Cependant, ses performances se dégradent bien entendu si l'on fait l'hypothèse que les processus du

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique